Exploring Airbnbs in New York City

San Francisco State University

Nancy Jaramillo and Banaz Sinjary

Math 448 - Dr. Tao He

May 6, 2025

1 Introduction

As my generation begins to explore the world on our own and with our friends, we are looking for the best places to stay in cities all over the world. Due to our struggling economy, hotel and restaurant prices have continued to skyrocket. This caused traveler like ourselves to turn to other options such as Airbnb. Airbnb is an online platform that allows users to rent their properties directly to consumers within seconds. A traveler can input their budget, desired location, and other specifications and receive plenty of options to rent based on their preferences. As my partner and I are avid Airbnb users, we decided to take a deep dive into what makes certain rentals desirable to consumers. We chose the data set on Airbnbs in New York City because it is a huge tourist destination for Americans and those traveling from abroad. We would like to see which features play significant roles in prices.

2 The Dataset

We used the OpenAirbnb dataset, which contains over 85,000 Airbnb listings in New York City, including variables such as price, room type, cancellation policy, host identity, availability, and more. The raw data included a mix of numeric, character, and categorical columns, some of which had inconsistent formatting or missing values.

To clean the data, we removed columns that weren't useful for modeling, like host names, host IDs, and unstructured text fields like house rules. We fixed formatting issues by removing dollar signs and commas from the price and service.fee columns and converting them into numeric values. We also standardized category entries. For example, a misspelled value "brookln" was corrected to "Brooklyn" in the neighbourhood.group column. Finally, we converted relevant columns to factors and dropped any rows with missing data to ensure model compatibility.

3 Model Selection

3.1 Linear Regression

To begin analyzing this data, we decided to run a linear regression model dependent on price. The independent factors used to run this model were service fee, room type, number of reviews, neighborhood group, calculated host listings count, cancellation policy, availability out of 365, construction year, instant bookable, minimum nights required, host identity verified, reviews per month, and review rate number. This linear regression model showed us that the only significant variable was the service fee. We then calculated the RMSE for this model and found it to be 1.414598. This means that the predictions have a low variation of about 1.4%.

3.2 Validation Set Approach

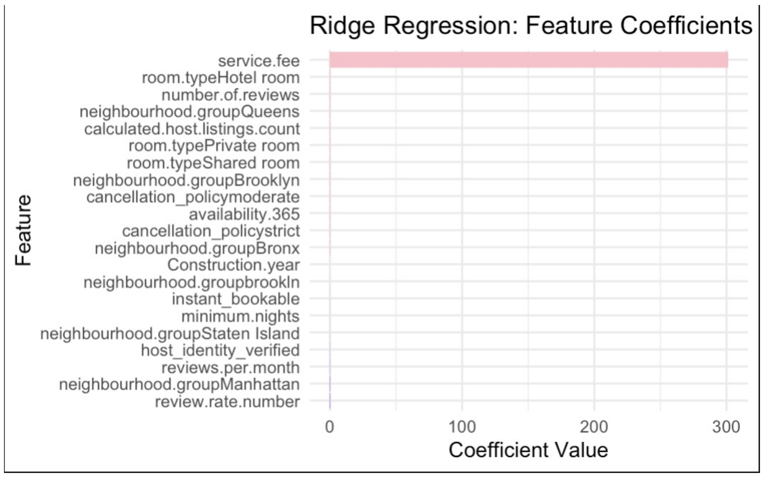
We decided to perform a validation set approach to give us an unbiased evaluation of how our model was performing. To do this, we started by splitting the data into 70-30 training and test data. We then performed a linear regression on a reduced model using only the features the original linear regression model found significant. We found the calculated RMSE to be 1.414129. This means that there is a variation of about 1.4% We repeated this process two more times but using a 60-40 split and a 80-20 split to see if there was a difference. With the 60-40 split, we calculated the RMSE to be 1.414568. With the 80-20 split, we calculated the RMSE to be 1.414286. These tests show a very small difference between the types of split as they all show about a 1.4% variation.

3.3 Ridge Regression

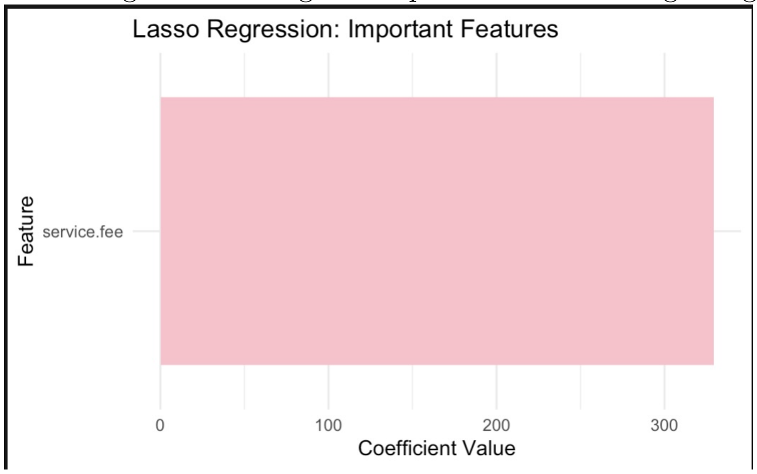
We decided to use Ridge Regression in an attempt to improve stability and accuracy when making predictions. In order to run this model, we set alpha to zero then split the data into training and test data. Upon running the model, we found the RMSE to be 34.14. With the mean and median of the price according the original Airbnb data being $625, the predictions made with ridge regression are about 5.4% off. This means that each prediction has a variation of about $34.14. As we know, ridge regression makes the insignificant coefficients small, but does not get rid of them completely. With this in mind, we wanted to see which features this model found insignificant. To do so we write the code and plotted the results.

3.4 LASSO

Next we used Least Absolute Shrinkage and Selection Operator, lasso. To run this model, we set alpha to 1 and split the data into training data and test data. After running the model, we



evaluated the performance and found the RMSE to be 16.2. This means that using this method, our predictions are about $16.20 off from the actual price. Given that the median and mean price of an Airbnb from out dataset is $625, lasso is about 2.6% off. Due to it's low RMSE, we decided to investigate which features this model found to be important. We did this by filtering out which coefficients that it sent to zero. The resulting important feature was service.fee. This report lines up with our original linear regression performed at the beginning of our analysis.



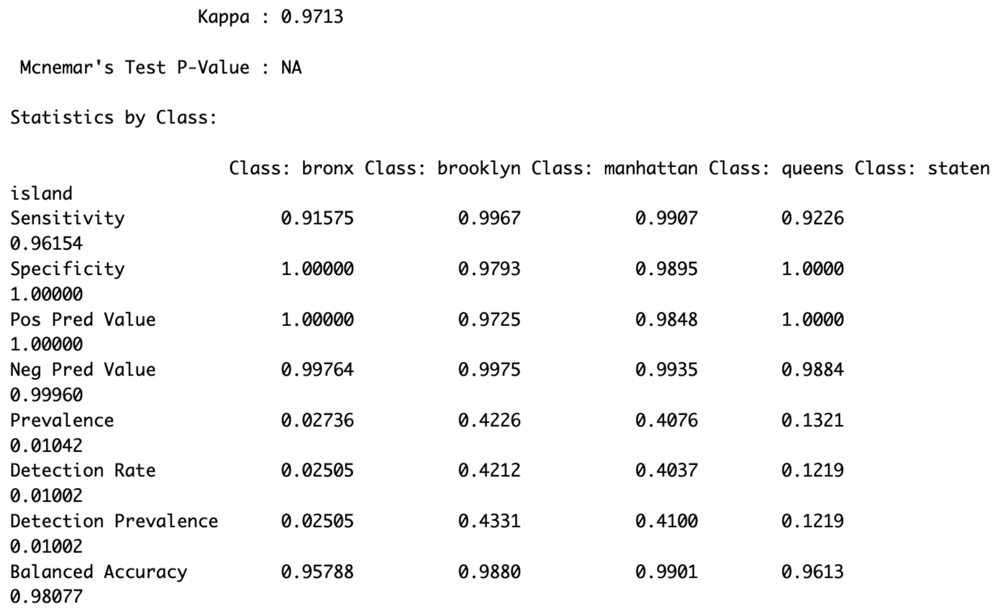
3.5 Random Forest

We used a Random Forest model to predict the neighborhood group of Airbnb listings based on various features such as price, service fee, room type, and cancellation policy. Random Forest is an ensemble learning method that builds multiple decision trees, each trained on a random subset of the data. It then combines the results of these trees to make more accurate predictions. The model works by making splits at each node of the tree based on the feature values that best separate the data, and this process is repeated across many trees to improve accuracy and reduce overfitting.

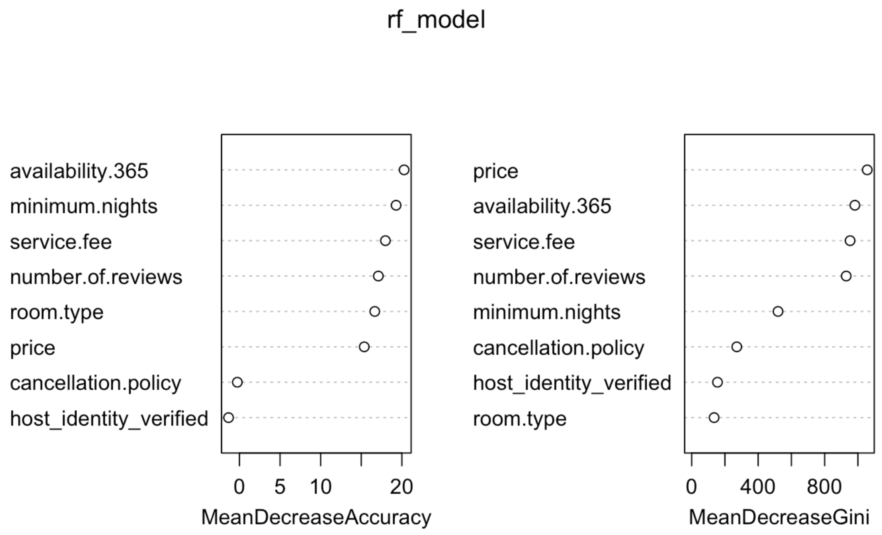
To implement Random Forest, we first preprocessed the data by converting categorical variables like room type and neighborhood group into dummy variables. Numeric features such as price and service fee were cleaned and scaled for consistency. We then trained the model on a subset of 10,000 data points, ensuring it was properly split into training and test sets. The Random Forest model allowed us to capture complex relationships between the features and predict the neighborhood group more accurately than simpler models. The final predictions were aggregated from the individual decision trees, providing an accurate output.

The confusion matrix shows the accuracy of the model in classifying the different neighborhood groups. The sensitivity values identify how often a true positive is predicted for each neighborhood. Specificity represents the identification of true negatives. The high values we see in both categories show that the model is performing very well at identifying correct classes.

The mean decrease accuracy, and mean decrease gini plots show features that make a difference in the decision making process. As we assumed, features like price, availability, and service fee were helpful in predicting which neighborhood the residence is located in. These features contribute the most to the models accuracy. Overall, the outputs confirm that the Random Forest model can accurately classify neighborhoods and define important features that make the most impact.



```
                  Kappa : 0.9713

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: bronx Class: brooklyn Class: manhattan Class: queens Class: staten
island
Sensitivity               0.91575          0.9967           0.9907        0.9226
0.96154
Specificity               1.00000          0.9793           0.9895        1.0000
1.00000
Pos Pred Value            1.00000          0.9725           0.9848        1.0000
1.00000
Neg Pred Value            0.99764          0.9975           0.9935        0.9884
0.99960
Prevalence                0.02736          0.4226           0.4076        0.1321
0.01042
Detection Rate            0.02505          0.4212           0.4037        0.1219
0.01002
Detection Prevalence      0.02505          0.4331           0.4100        0.1219
0.01002
Balanced Accuracy         0.95788          0.9880           0.9901        0.9613
0.98077
```



3.6 K-fold Validation

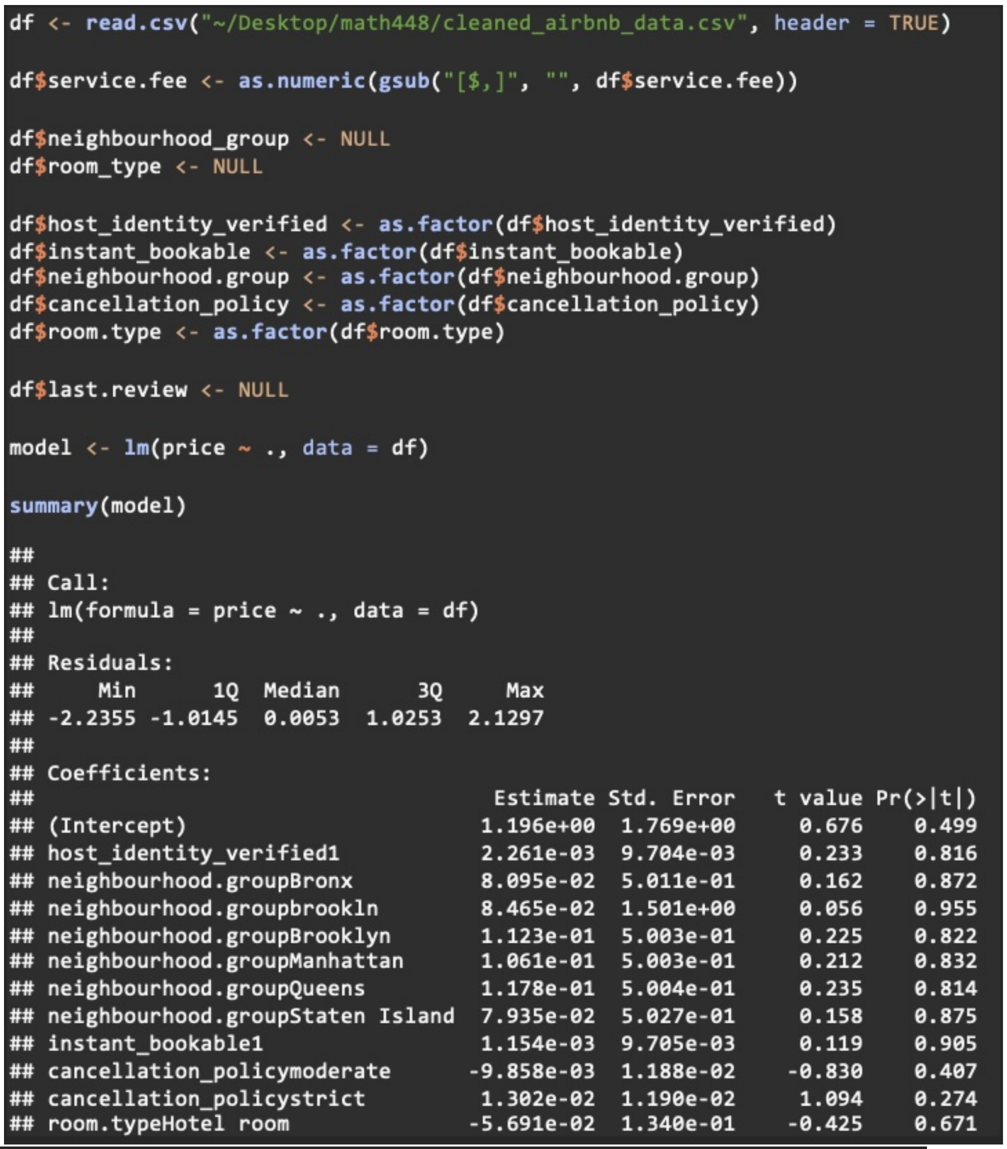
The K-Nearest Neighbors (KNN) algorithm was used to predict the price listings for Airbnbs based on numerical features. Somethings we took into account were service fee, room type, and number of reviews. KNN is a non parametric algorithm. Instead of focusing on input/output relationships, it makes predictions by finding the k closest data points to the test sample and averaging the values. For this model we chose k = 5, this means each prediction is based on an average of the 5 nearest neighbors.

To evaluate the results we used the Root Mean Squared Error (RMSE) which was 63.78. This means on average the predictions were off by about 63.78 dollars. We also used the Mean Absolute Error (MAE), it was 53.88 meaning the models predictions were off by 53.88 units on average. Then we calculated the Mean Absolute Percentage Error (MAPE) which was 14.09. From this we can conclude that predictions were about 14.09 percent off actual values. Finally the model's R squared value of 0.9631 shows that 96.31 percent of the variance in price can be explained by the KNN model. These results suggest that the KNN model works quite well overall, but could do with some more parameter tuning for future work.

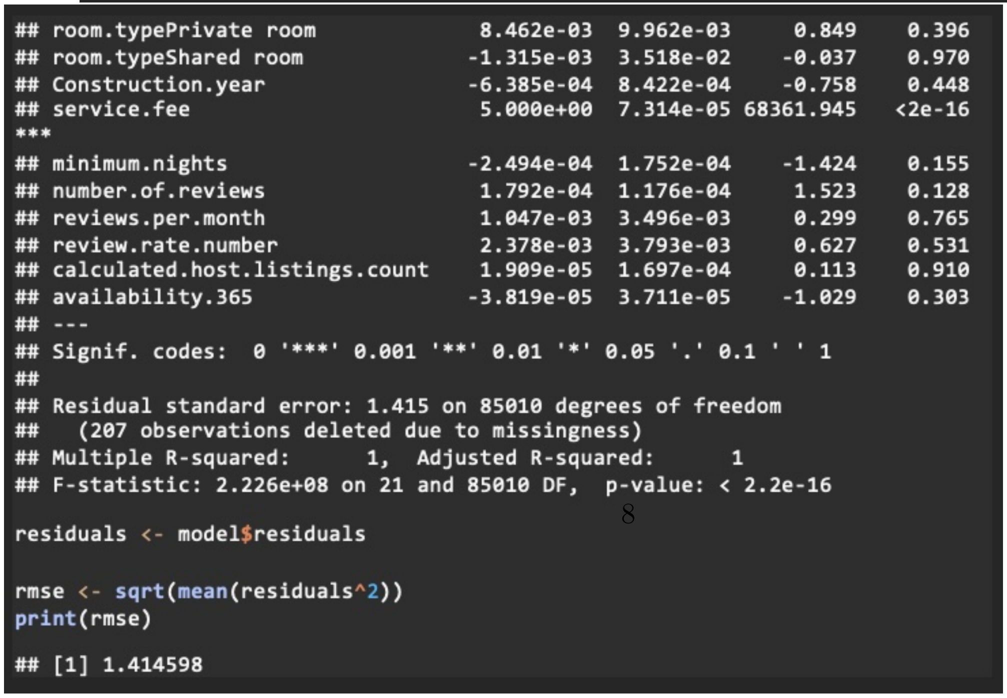
4 Conclusion

In this analysis we explored different machine learning models to predict price and relating factors. These predictions were based on various features drawn from the OpenAirbnb dataset. We began by cleaning and prepossessing the data, addressing missing values and spelling errors. Irrelevant categories were dropped as well to efficiently manage run time. After cleaning and normalizing the data we applied several data mining models such as, linear regression, ridge regression, LASSO, random forest, and k-nearest neighbors. Each model had its own rate of accuracy that defined its predictive methods. The random forest, LASSO, and KNN demonstrated very strong performance. Overall, the study shows how different machine learning algorithms work on making predictions with the same dataset. These predictions based on complex real world data show the complexity and relevance of data mining architectures.

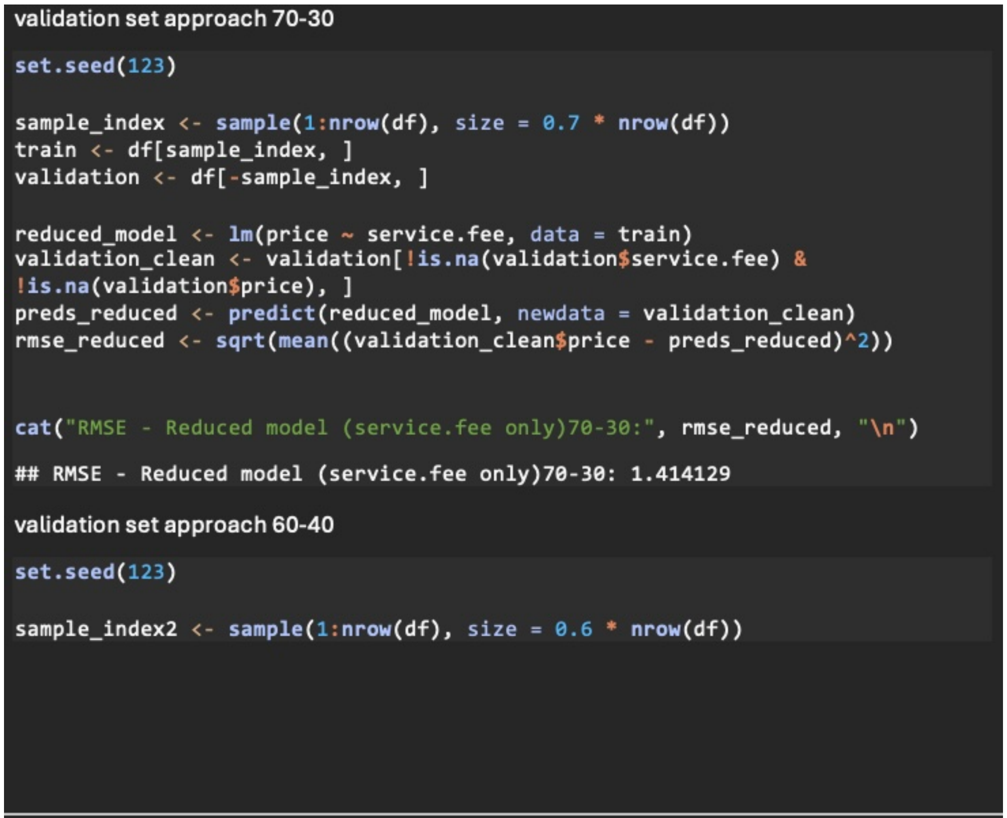
5 Appendix



```
df <- read.csv("~/Desktop/math448/cleaned_airbnb_data.csv", header = TRUE)

df$service.fee <- as.numeric(gsub("[$,]", "", df$service.fee))

df$neighbourhood_group <- NULL
df$room_type <- NULL

df$host_identity_verified <- as.factor(df$host_identity_verified)
df$instant_bookable <- as.factor(df$instant_bookable)
df$neighbourhood.group <- as.factor(df$neighbourhood.group)
df$cancellation_policy <- as.factor(df$cancellation_policy)
df$room.type <- as.factor(df$room.type)

df$last.review <- NULL

model <- lm(price ~ ., data = df)

summary(model)

##
## Call:
## lm(formula = price ~ ., data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2355 -1.0145  0.0053  1.0253  2.1297
##
## Coefficients:
##                                  Estimate Std. Error   t value Pr(>|t|)
## (Intercept)                     1.196e+00  1.769e+00     0.676    0.499
## host_identity_verified1         2.261e-03  9.704e-03     0.233    0.816
## neighbourhood.groupBronx        8.095e-02  5.011e-01     0.162    0.872
## neighbourhood.groupbrookln      8.465e-02  1.501e+00     0.056    0.955
## neighbourhood.groupBrooklyn     1.123e-01  5.003e-01     0.225    0.822
## neighbourhood.groupManhattan    1.061e-01  5.003e-01     0.212    0.832
## neighbourhood.groupQueens       1.178e-01  5.004e-01     0.235    0.814
## neighbourhood.groupStaten Island 7.935e-02  5.027e-01     0.158    0.875
## instant_bookable1               1.154e-03  9.705e-03     0.119    0.905
## cancellation_policymoderate    -9.858e-03  1.188e-02    -0.830    0.407
## cancellation_policystrict       1.302e-02  1.190e-02     1.094    0.274
## room.typeHotel room            -5.691e-02  1.340e-01    -0.425    0.671
```



```
## room.typePrivate room           8.462e-03  9.962e-03     0.849    0.396
## room.typeShared room           -1.315e-03  3.518e-02    -0.037    0.970
## Construction.year              -6.385e-04  8.422e-04    -0.758    0.448
## service.fee                     5.000e+00  7.314e-05 68361.945   <2e-16
***
## minimum.nights                 -2.494e-04  1.752e-04    -1.424    0.155
## number.of.reviews               1.792e-04  1.176e-04     1.523    0.128
## reviews.per.month               1.047e-03  3.496e-03     0.299    0.765
## review.rate.number              2.378e-03  3.793e-03     0.627    0.531
## calculated.host.listings.count  1.909e-05  1.697e-04     0.113    0.910
## availability.365               -3.819e-05  3.711e-05    -1.029    0.303
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.415 on 85010 degrees of freedom
##   (207 observations deleted due to missingness)
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 2.226e+08 on 21 and 85010 DF,  p-value: < 2.2e-16

residuals <- model$residuals

rmse <- sqrt(mean(residuals^2))
print(rmse)

## [1] 1.414598
```

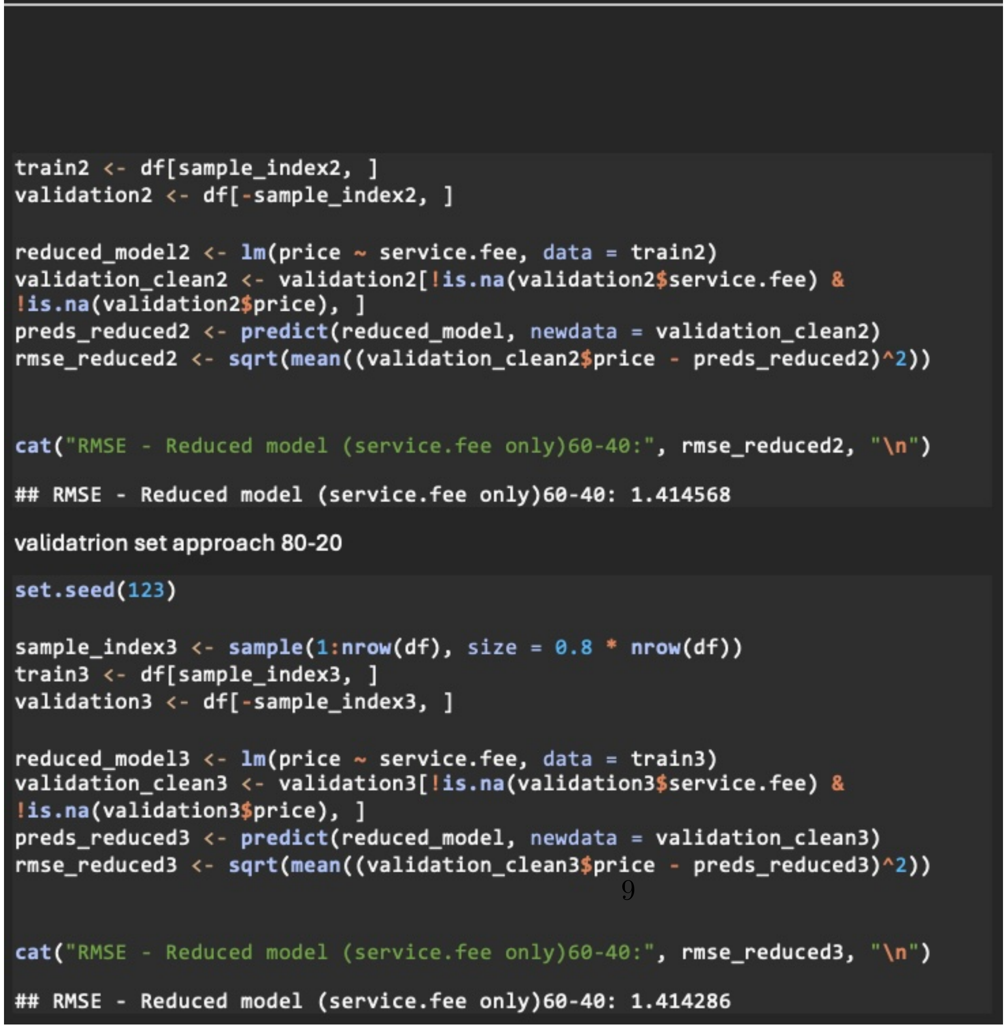


validation set approach 70-30

```
set.seed(123)

sample_index <- sample(1:nrow(df), size = 0.7 * nrow(df))
train <- df[sample_index, ]
validation <- df[-sample_index, ]

reduced_model <- lm(price ~ service.fee, data = train)
validation_clean <- validation[!is.na(validation$service.fee) &
!is.na(validation$price), ]
preds_reduced <- predict(reduced_model, newdata = validation_clean)
rmse_reduced <- sqrt(mean((validation_clean$price - preds_reduced)^2))


cat("RMSE - Reduced model (service.fee only)70-30:", rmse_reduced, "\n")

## RMSE - Reduced model (service.fee only)70-30: 1.414129
```

validation set approach 60-40

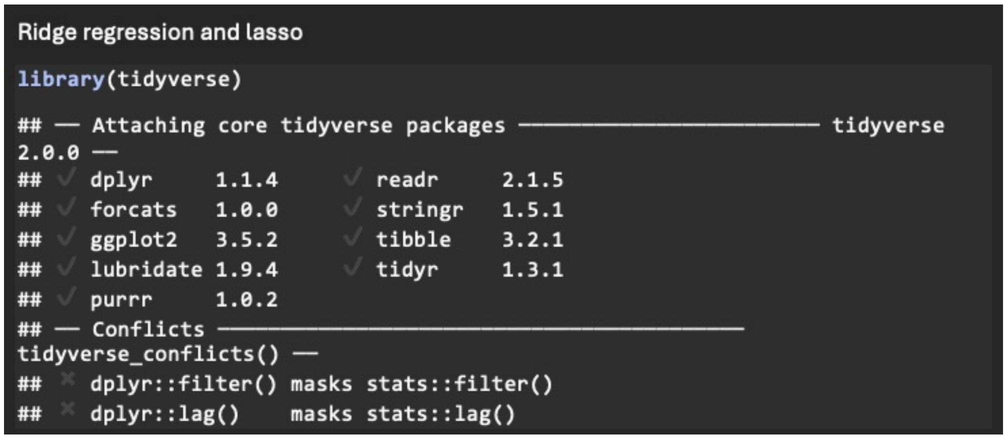
```
set.seed(123)

sample_index2 <- sample(1:nrow(df), size = 0.6 * nrow(df))
```



```
train2 <- df[sample_index2, ]
validation2 <- df[-sample_index2, ]

reduced_model2 <- lm(price ~ service.fee, data = train2)
validation_clean2 <- validation2[!is.na(validation2$service.fee) &
!is.na(validation2$price), ]
preds_reduced2 <- predict(reduced_model, newdata = validation_clean2)
rmse_reduced2 <- sqrt(mean((validation_clean2$price - preds_reduced2)^2))


cat("RMSE - Reduced model (service.fee only)60-40:", rmse_reduced2, "\n")

## RMSE - Reduced model (service.fee only)60-40: 1.414568
```

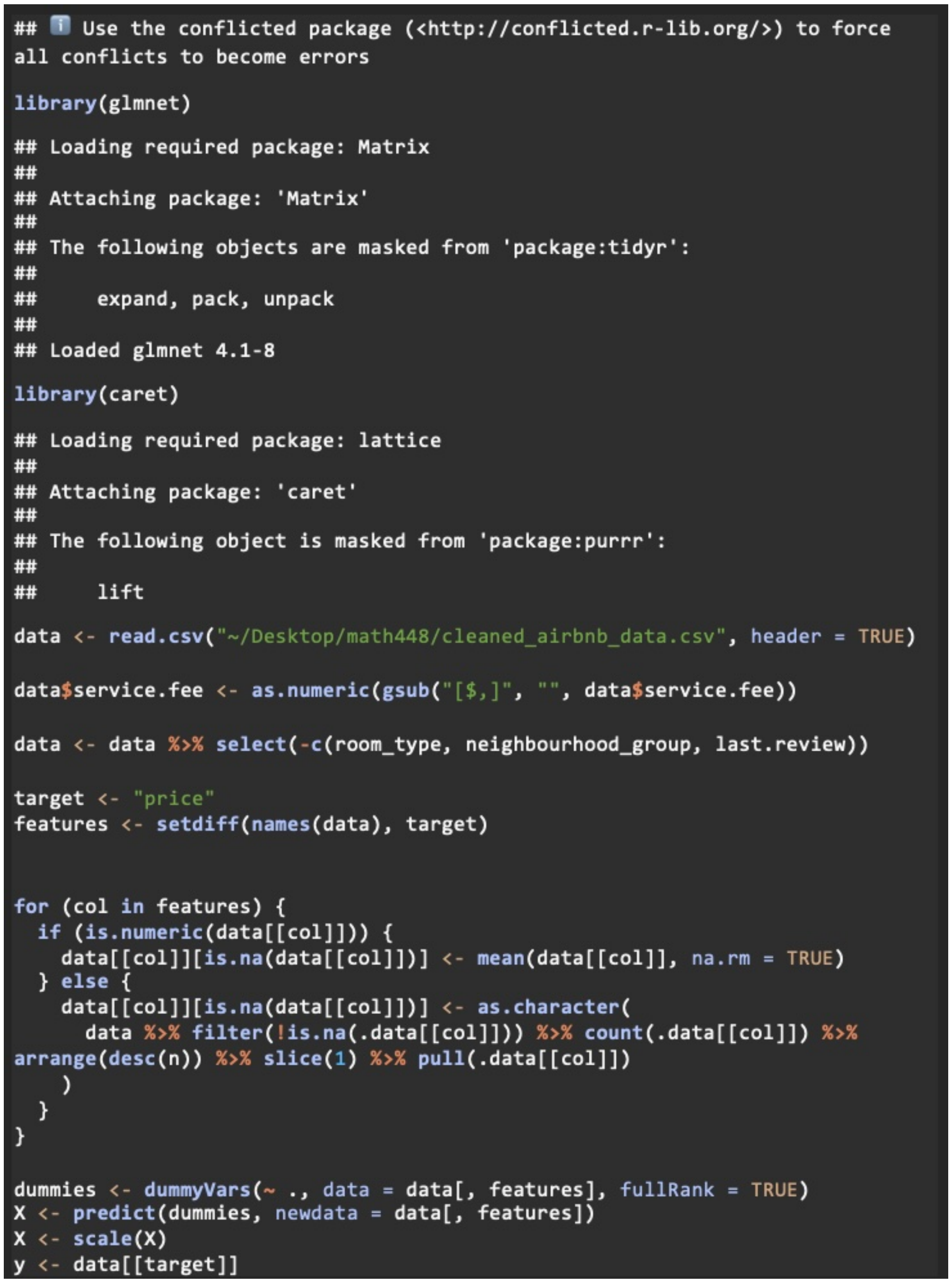
validatrion set approach 80-20

```
set.seed(123)

sample_index3 <- sample(1:nrow(df), size = 0.8 * nrow(df))
train3 <- df[sample_index3, ]
validation3 <- df[-sample_index3, ]

reduced_model3 <- lm(price ~ service.fee, data = train3)
validation_clean3 <- validation3[!is.na(validation3$service.fee) &
!is.na(validation3$price), ]
preds_reduced3 <- predict(reduced_model, newdata = validation_clean3)
rmse_reduced3 <- sqrt(mean((validation_clean3$price - preds_reduced3)^2))


cat("RMSE - Reduced model (service.fee only)60-40:", rmse_reduced3, "\n")

## RMSE - Reduced model (service.fee only)60-40: 1.414286
```

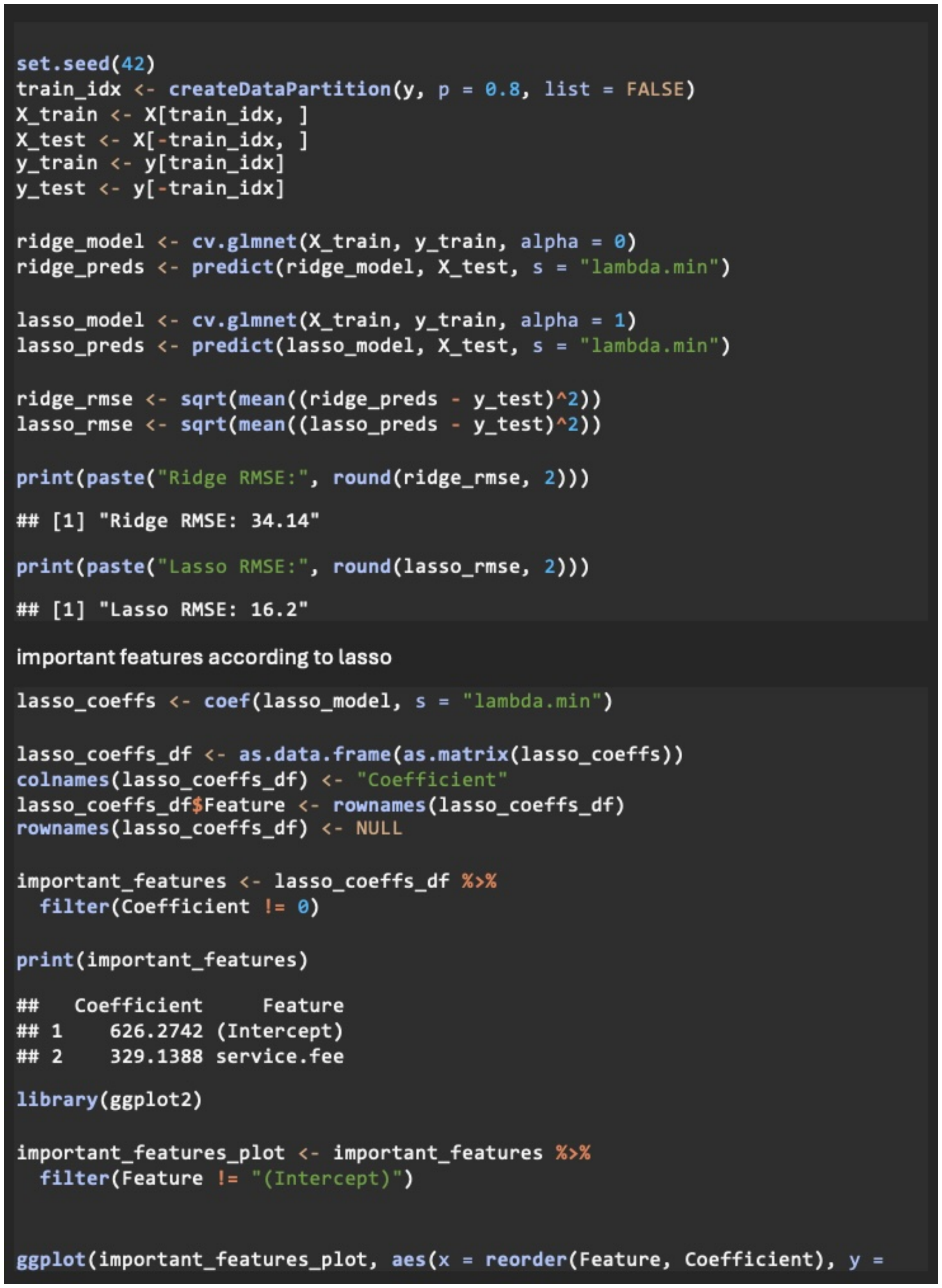

Ridge regression and lasso

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.2     ✓ tibble    3.2.1
## ✓ lubridate 1.9.4     ✓ tidyr     1.3.1
## ✓ purrr     1.0.2
## ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```



```
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force
all conflicts to become errors

library(glmnet)

## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8

library(caret)

## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift

data <- read.csv("~/Desktop/math448/cleaned_airbnb_data.csv", header = TRUE)

data$service.fee <- as.numeric(gsub("[$,]", "", data$service.fee))

data <- data %>% select(-c(room_type, neighbourhood_group, last.review))

target <- "price"
features <- setdiff(names(data), target)


for (col in features) {
  if (is.numeric(data[[col]])) {
    data[[col]][is.na(data[[col]])] <- mean(data[[col]], na.rm = TRUE)
  } else {
    data[[col]][is.na(data[[col]])] <- as.character(
      data %>% filter(!is.na(.data[[col]])) %>% count(.data[[col]]) %>%
arrange(desc(n)) %>% slice(1) %>% pull(.data[[col]])
    )
  }
}

dummies <- dummyVars(~ ., data = data[, features], fullRank = TRUE)
X <- predict(dummies, newdata = data[, features])
X <- scale(X)
y <- data[[target]]
```

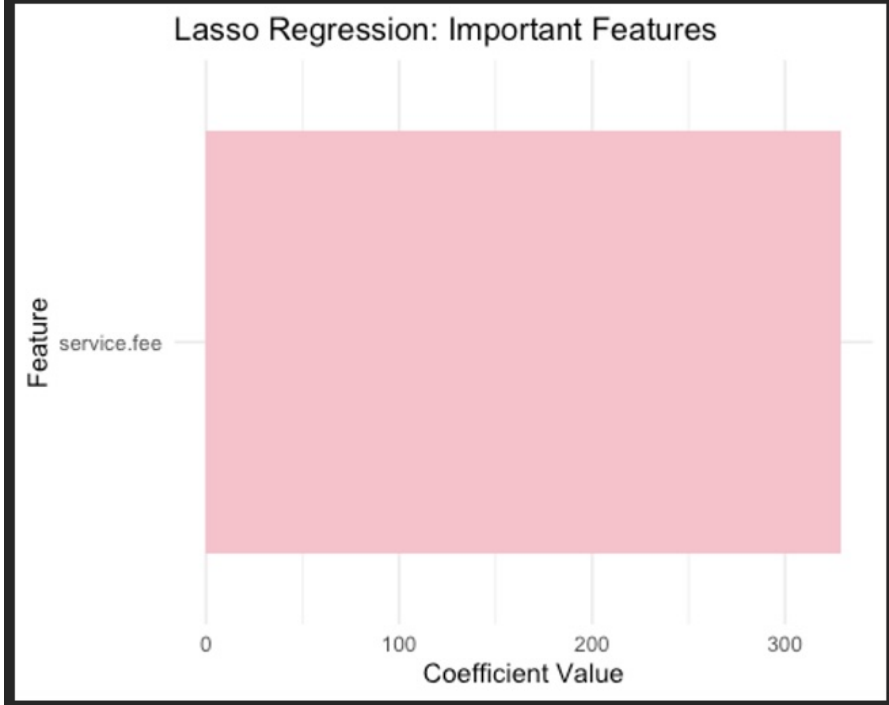


```r
set.seed(42)
train_idx <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X[train_idx, ]
X_test <- X[-train_idx, ]
y_train <- y[train_idx]
y_test <- y[-train_idx]

ridge_model <- cv.glmnet(X_train, y_train, alpha = 0)
ridge_preds <- predict(ridge_model, X_test, s = "lambda.min")

lasso_model <- cv.glmnet(X_train, y_train, alpha = 1)
lasso_preds <- predict(lasso_model, X_test, s = "lambda.min")

ridge_rmse <- sqrt(mean((ridge_preds - y_test)^2))
lasso_rmse <- sqrt(mean((lasso_preds - y_test)^2))

print(paste("Ridge RMSE:", round(ridge_rmse, 2)))

## [1] "Ridge RMSE: 34.14"

print(paste("Lasso RMSE:", round(lasso_rmse, 2)))

## [1] "Lasso RMSE: 16.2"
```

important features according to lasso

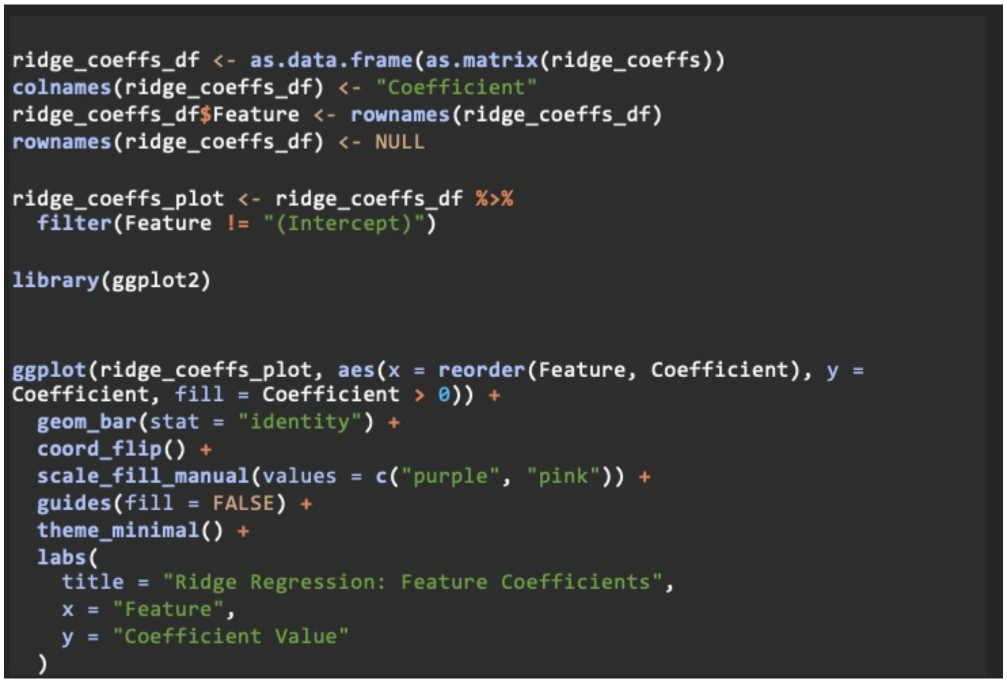
```r
lasso_coeffs <- coef(lasso_model, s = "lambda.min")

lasso_coeffs_df <- as.data.frame(as.matrix(lasso_coeffs))
colnames(lasso_coeffs_df) <- "Coefficient"
lasso_coeffs_df$Feature <- rownames(lasso_coeffs_df)
rownames(lasso_coeffs_df) <- NULL

important_features <- lasso_coeffs_df %>%
  filter(Coefficient != 0)

print(important_features)

##   Coefficient     Feature
## 1    626.2742 (Intercept)
## 2    329.1388 service.fee

library(ggplot2)

important_features_plot <- important_features %>%
  filter(Feature != "(Intercept)")


ggplot(important_features_plot, aes(x = reorder(Feature, Coefficient), y =
Coefficient, fill = Coefficient > 0)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  scale_fill_manual(values = c("pink", "purple")) +
  guides(fill = FALSE) +
  theme_minimal() +
  labs(
    title = "Lasso Regression: Important Features",
    x = "Feature",
    y = "Coefficient Value"
  )
```

```
## Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use
"none" instead as
## of ggplot2 3.3.4.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

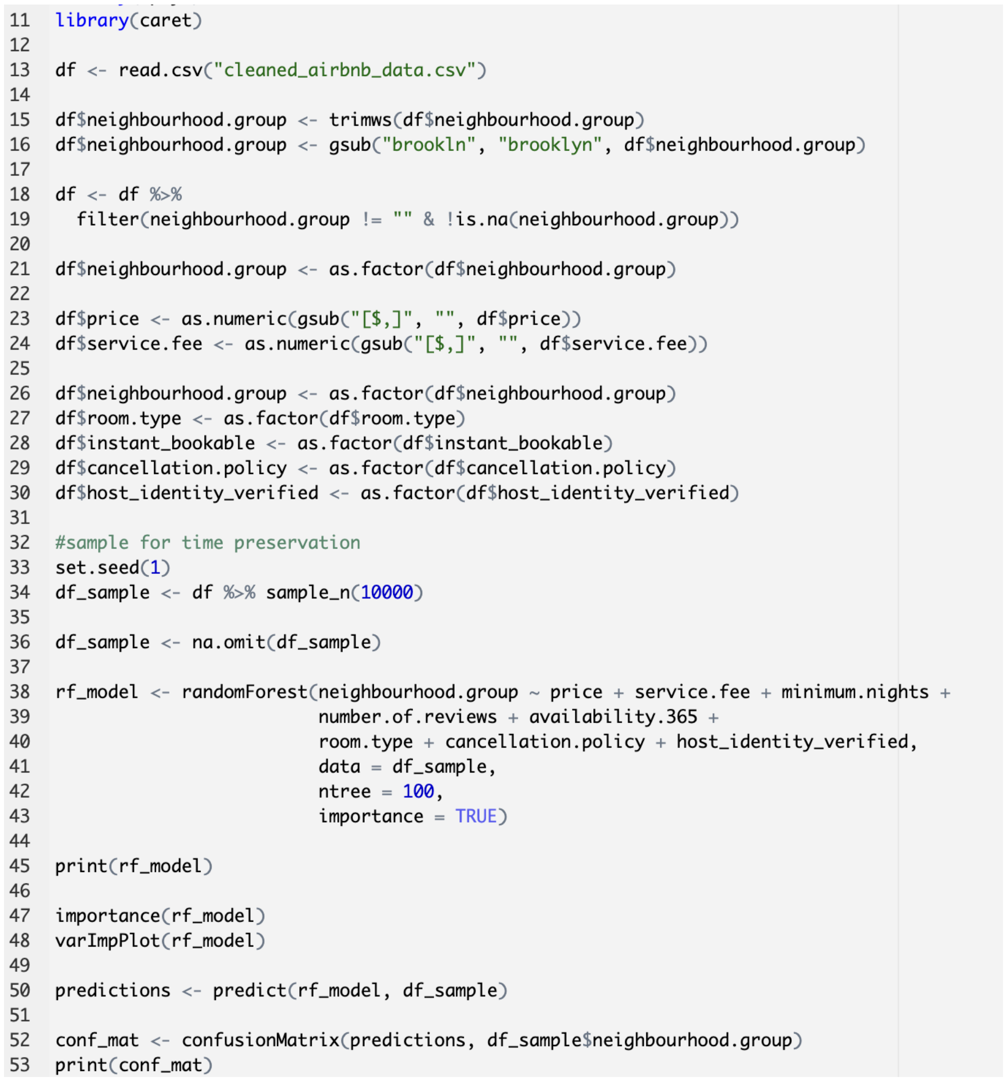


important features according to ridge

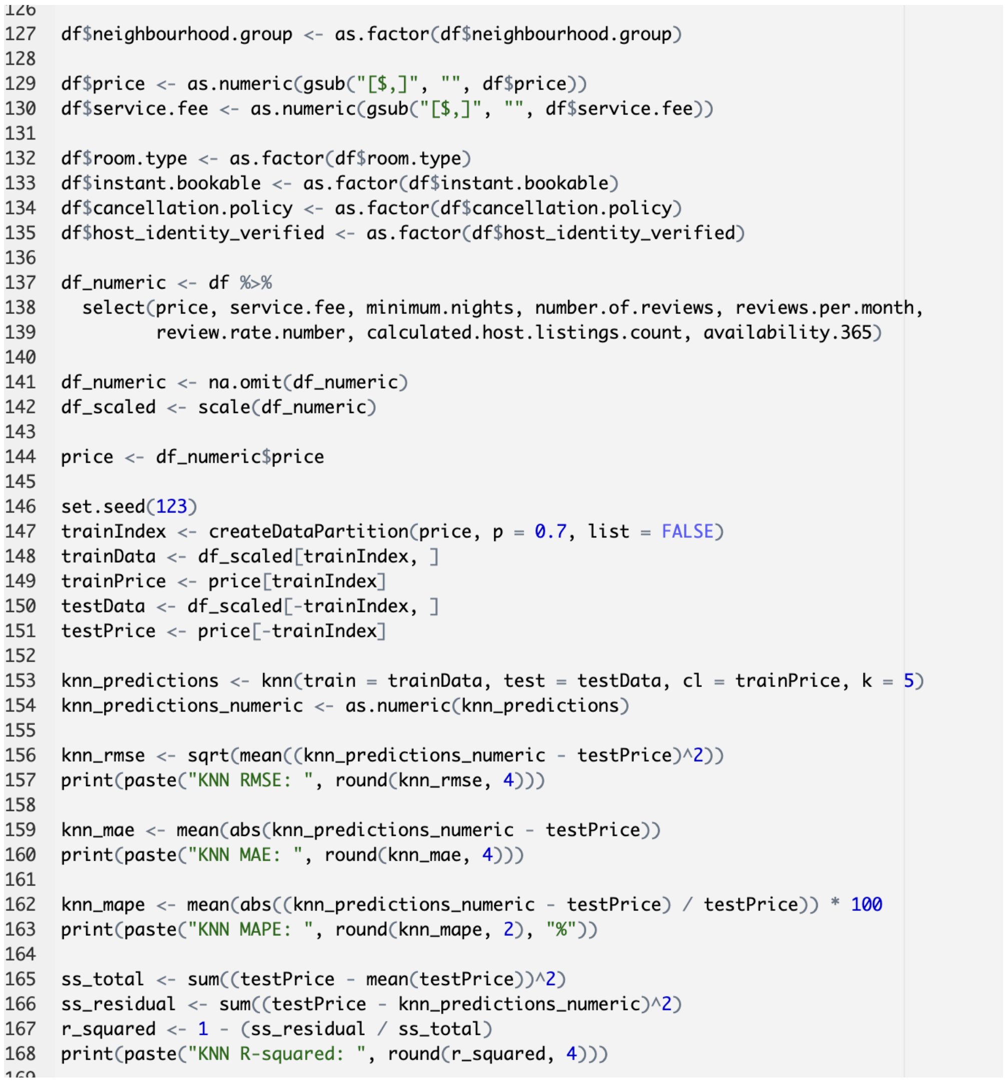
```r
library(ggplot2)
ridge_coeffs <- coef(ridge_model, s = "lambda.min")
```



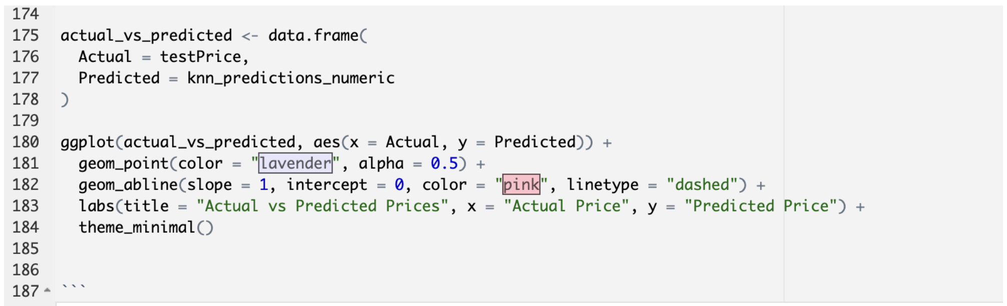
```r
ridge_coeffs_df <- as.data.frame(as.matrix(ridge_coeffs))
colnames(ridge_coeffs_df) <- "Coefficient"
ridge_coeffs_df$Feature <- rownames(ridge_coeffs_df)
rownames(ridge_coeffs_df) <- NULL

ridge_coeffs_plot <- ridge_coeffs_df %>%
  filter(Feature != "(Intercept)")

library(ggplot2)


ggplot(ridge_coeffs_plot, aes(x = reorder(Feature, Coefficient), y =
Coefficient, fill = Coefficient > 0)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  scale_fill_manual(values = c("purple", "pink")) +
  guides(fill = FALSE) +
  theme_minimal() +
  labs(
    title = "Ridge Regression: Feature Coefficients",
    x = "Feature",
    y = "Coefficient Value"
  )
```

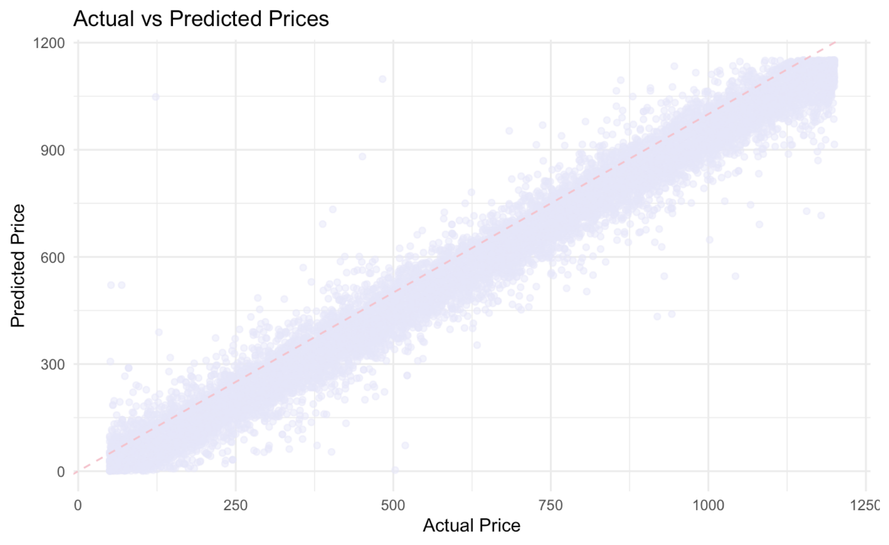


```r
library(caret)

df <- read.csv("cleaned_airbnb_data.csv")

df$neighbourhood.group <- trimws(df$neighbourhood.group)
df$neighbourhood.group <- gsub("brookln", "brooklyn", df$neighbourhood.group)

df <- df %>%
  filter(neighbourhood.group != "" & !is.na(neighbourhood.group))

df$neighbourhood.group <- as.factor(df$neighbourhood.group)

df$price <- as.numeric(gsub("[$,]", "", df$price))
df$service.fee <- as.numeric(gsub("[$,]", "", df$service.fee))

df$neighbourhood.group <- as.factor(df$neighbourhood.group)
df$room.type <- as.factor(df$room.type)
df$instant_bookable <- as.factor(df$instant_bookable)
df$cancellation.policy <- as.factor(df$cancellation.policy)
df$host_identity_verified <- as.factor(df$host_identity_verified)

#sample for time preservation
set.seed(1)
df_sample <- df %>% sample_n(10000)

df_sample <- na.omit(df_sample)

rf_model <- randomForest(neighbourhood.group ~ price + service.fee + minimum.nights +
                         number.of.reviews + availability.365 +
                         room.type + cancellation.policy + host_identity_verified,
                         data = df_sample,
                         ntree = 100,
                         importance = TRUE)

print(rf_model)

importance(rf_model)
varImpPlot(rf_model)

predictions <- predict(rf_model, df_sample)

conf_mat <- confusionMatrix(predictions, df_sample$neighbourhood.group)
print(conf_mat)
```



```r
df$neighbourhood.group <- as.factor(df$neighbourhood.group)

df$price <- as.numeric(gsub("[$,]", "", df$price))
df$service.fee <- as.numeric(gsub("[$,]", "", df$service.fee))

df$room.type <- as.factor(df$room.type)
df$instant.bookable <- as.factor(df$instant.bookable)
df$cancellation.policy <- as.factor(df$cancellation.policy)
df$host_identity_verified <- as.factor(df$host_identity_verified)

df_numeric <- df %>%
  select(price, service.fee, minimum.nights, number.of.reviews, reviews.per.month,
         review.rate.number, calculated.host.listings.count, availability.365)

df_numeric <- na.omit(df_numeric)
df_scaled <- scale(df_numeric)

price <- df_numeric$price

set.seed(123)
trainIndex <- createDataPartition(price, p = 0.7, list = FALSE)
trainData <- df_scaled[trainIndex, ]
trainPrice <- price[trainIndex]
testData <- df_scaled[-trainIndex, ]
testPrice <- price[-trainIndex]

knn_predictions <- knn(train = trainData, test = testData, cl = trainPrice, k = 5)
knn_predictions_numeric <- as.numeric(knn_predictions)

knn_rmse <- sqrt(mean((knn_predictions_numeric - testPrice)^2))
print(paste("KNN RMSE: ", round(knn_rmse, 4)))

knn_mae <- mean(abs(knn_predictions_numeric - testPrice))
print(paste("KNN MAE: ", round(knn_mae, 4)))

knn_mape <- mean(abs((knn_predictions_numeric - testPrice) / testPrice)) * 100
print(paste("KNN MAPE: ", round(knn_mape, 2), "%"))

ss_total <- sum((testPrice - mean(testPrice))^2)
ss_residual <- sum((testPrice - knn_predictions_numeric)^2)
r_squared <- 1 - (ss_residual / ss_total)
print(paste("KNN R-squared: ", round(r_squared, 4)))
```



```r

actual_vs_predicted <- data.frame(
  Actual = testPrice,
  Predicted = knn_predictions_numeric
)

ggplot(actual_vs_predicted, aes(x = Actual, y = Predicted)) +
  geom_point(color = "lavender", alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "pink", linetype = "dashed") +
  labs(title = "Actual vs Predicted Prices", x = "Actual Price", y = "Predicted Price") +
  theme_minimal()


```
```


```